PARTITIONING-BASED PLACEMENT

Jing Lee

Department of Electronic Engineering

Southern Taiwan University of Technology

Tainan, Taiwan 701, R.O.C.

Email: leejing@mail.stut.edu.tw

Partitioning plays a key role in the design of a computer system in general and VLSI chips. Any complex system, consisting of a large number of components, cannot be designed efficiently without decomposing it into a set of smaller subsystems [Nie83].

Placement algorithms based on partitioning divide components into two or more partitions while reserving space (called a **block**) for the components. This process of dividing blocks into smaller blocks and partitioning components into the reduced blocks continues until the number of components per block is small. These algorithms are widely used in modern layout systems [Bre77].

Since the partitioning problem has been proven to be **NP-complete** [Gar79], it is impractical to perform an exhaustive search to find the optimal partition for a large network. Thus, algorithms based on heuristic rationales are commonly employed. There are basically two kinds of heuristic algorithms, **constructive algorithms** and **iterative improvement algorithms**, for this problem. Cnstructive algorithms often construct a partition using heuristic rules in a sequential, deterministic manner. The goal of iterative improvement algorithms is to improve upon an existing partition. An ideal approach would be a combination of the two. First, one gets an initial partitioning solution using a constructive algorithm, and then uses an iterative improvement algorithm to improve it. It should be pointed out that a better initial partitioning solution always leads to a better solution after iterative improvement. Also the larger a network, the more important is the initial constructive problem [Han76, Gro87], so it is desirable to develop a better constructive algorithm.

Clustering algorithms are usually used to construct an initial partitioning [Sch72, Cox80, Sch82, Gro87, Ng87]. Traditional clustering algorithms use a **graph** to represent a network. These algorithms are easy to implement and very fast. A major error of these algorithms is the **edge exaggeration** that occurred in the transformation of the multi-pin nets [Sch72]. Different weighting functions as described in Chapter 2 are used to modify the edge exaggeration.

Another type of algorithms are based on hypergraph model. Several partitioning heuristics based on **hypergraph** model are described in [Sch72, Bre77, Kri84, Don88, Che92, Fid92, Hag92]. These algorithms are **top-down**, **iterative** approaches. This top-down design approach considers higher levels of abstraction before it considers more detailed levels and tends to avoid heavy wiring congestion typically found in the center of the layout surface. For most cases, top-down approaches give better solutions than bottom-up approaches. However, these top-down approaches are restricted that all vertices must be even sizes or dissipated heat. So, for the cases of power hybrid circuits or MCM chips that some components have higher heat generation than others, these top-down approaches are improper. Additional characteristics of the top-

down approaches are that the preplaced components are difficult (if not impossible) to handle and the resulting partitions can be grossly unequal, and so can not be easily processed. In addition, they are more computationally expensive than clustering algorithms [Mur80].

In this study, a new clustering algorithm is proposed for the networks partitioning problems. This algorithm is based on bottom-up approach, so it is very fast. Otherwise, it models networks with a hypergraph as top-down approaches, so the solutions obtained are much better than the traditional clustering algorithms. Otherwise, the algorithm can handle the problems with various options such as different sizes and heat dissipation. Therefore, it can be used to the power hybrid circuit and MCM chip designs. These applications will be introduced in the Chapter 5.

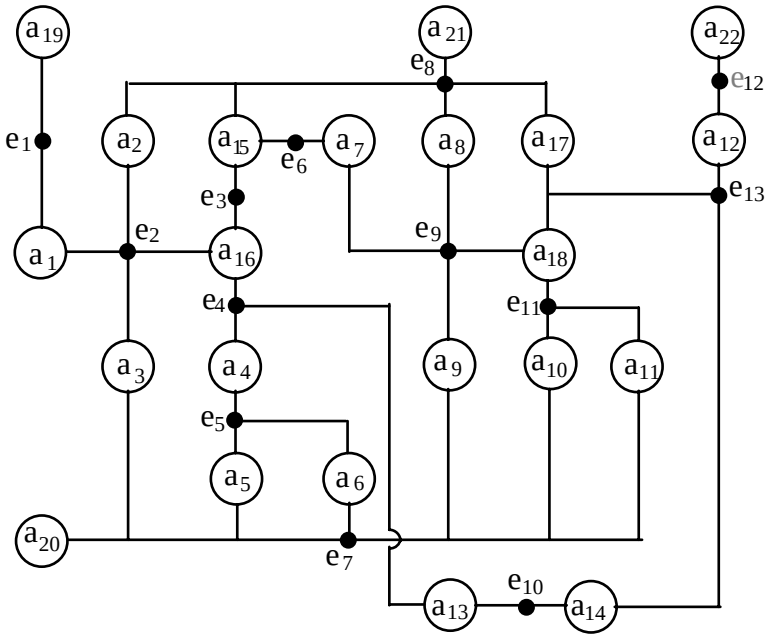
3.1 **Clustering Growth Algorithm**

In this method, a hypergraph is used to model networks. The **capacity** of a cluster is the number of atoms that the cluster can have. A cluster is **saturated** if its size is equal to its capacity, and **unsaturated** if its size is less than its capacity.

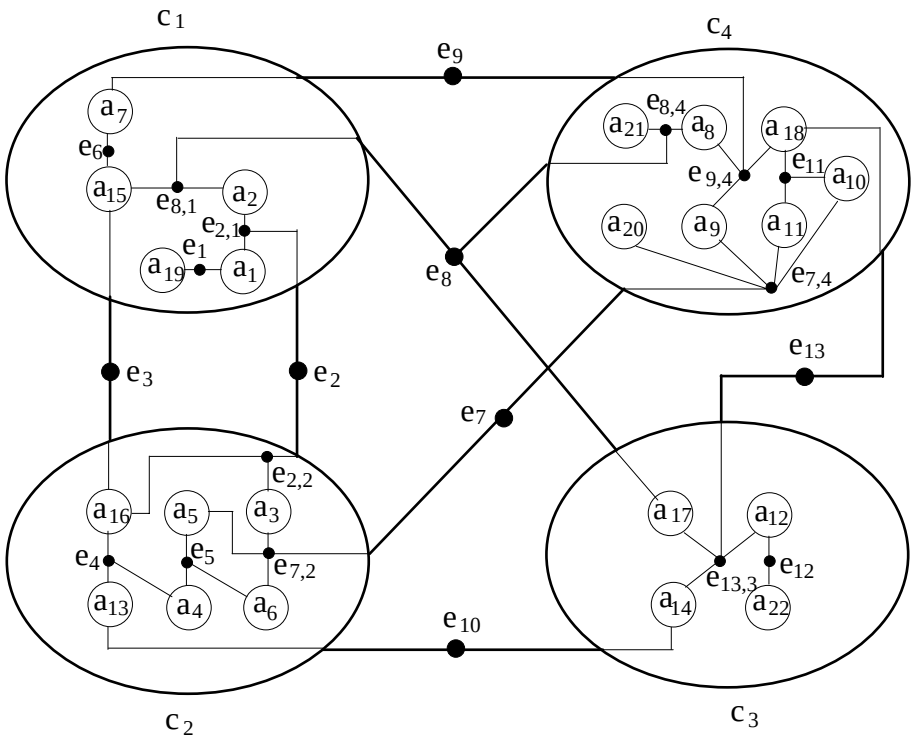
The clustering growth algorithm is outlined below:

procedure CLUSTERING(H, n)
//Input H; output the shunk hypergraph and n sub-hypergraphs, H_1, H_2,..., and H_n.//
(c_1, ..., c_n) ← SEEDS(H) //Select the seeds of clusters//
for m ← 1 **to** n **do** ASSIGN(c_m, H_m) **repeat** //Assign seeds into sub-hypergraphs//
while |V| > n **do**
    a_c ← SELECT_ATOM(A) //Select a_c in A = V–C. C = {c_1, ..., c_n}//
    c_m ← SELECT_CLUSTER(a_c, C) //Select c_m in C//
    ASSIGN(a_c, H_m) //Assign a_c and E(a_c) into H_m//
    HYPERGRAPH_CONTRACTION(a_c, c_m, H)
repeat
for m ← 1 **to** n **do** E_m ← E_m – {e_d | |V(e_d)| = 1} **repeat** //Delete one-vertex nets//
OUTPUT(H, H_1, ..., H_n)
end CLUSTERING

All vertices are atoms in the primary hypergraph. The clustering algorithm starts with selecting the seeds of clusters. There are n seeds for n-way partitioning of a hypergraph. Then, each sub-hypergraph, H_m, is initialized by including the seed, c_m. After that, the to-be-assigned atoms are selected, one at a time. Then, the selected atom, named **candidate** a_c, belongs to a cluster, called **merged cluster** c_m. Finally, a_c is assigned into H_m and the previous hypergraph is contracted to a one-vertex-less hypergraph. The above procedure is repeated until all atoms are assigned into the clusters. At last, the primary hypergraph is



(a) A hypergraph with 22 vertices and 13 nets.



(b) The shunk hypergraph and four sub-hypergraphs after executing the clustering algorithm.

Fig. 3.1 An example for clustering algorithm.

reduced to a shunk hypergraph, where each cluster is a sub-hypergraph. The objection of each assignment is subjected to reduce the complexity of the shunk hypergraph as more as possible. The complexity reduction can be measure by the number of the hyperedges deleted from the previous hypergraph.

An example is shown in Figs. 3.1(a) and (b). Fig. 3.1(a) is the primary hypergraph and Fig. 3.1(b) shows the shunk hypergraph with four sub-hypergraphs after executing our clustering algorithm from the seeds of a_{15}, a_{16}, a_{17}, and a_{18}. The shunk hypergraph is V = {c_1, c_2, c_3, c_4} and E = {e_2, e_3, e_7, e_8, e_9, e_{10}, e_{13}}. The sub-hypergraph, H_m, is the hypergraph inside the cluster, c_m. In Fig. 3.1(b), one can see that a hyperedge in Fig. 3.1(a) can be divided into several parts, where $e_{d,m}$ is a partial hyperedge of e_d, which is inside H_m. For example, $V(e_8)$ in Fig. 3.1(a) is the set of {a_2, a_8, a_{15}, a_{17}, a_{21}}. The $V(e_8)$ is divided into three sub-sets, namely, $V(e_{8,1})$ = {a_2, a_{15}}, $V(e_{8,4})$ = {a_8, a_{21}}, and $V(e_8)$ = {c_1, c_3, c_4} in Fig. 3.1(b).

Five main functions used in the clustering algorithm are described in the following.

3.1.1 **Function of SEEDS(H)**

The quality of partitioning results depends very much on the seeds. Thus, a good seed generation is important. However, there are few seed generators in the literature. Mostly, seeds are decided by the user.

We have looked into seed decision processes presented by skilled designers. We find that they choose seeds according to the following rules.

1. Seeds must be uniformly distributed throughout the hypergraph.
2. A heavily connected vertex that is adjacent to more neighbors is prior chosen as a seed.

In this study, a **seed generator** with the above characteristics is proposed. Before describing the procedure of the seed generator, we have to introduce the idea of vertex depth, which is used to decide the relative distances among seeds.

3.1.1.1 **Vertex Depth**

In the phase, the vertex set V will be partitioned into n disjoint sets V^i, $1 \le i \le k$. The set of V^1 has only one vertex, called initial vertex, and is denoted as v_{in}. The initial vertex is the lightest vertex neighboring the fewest vertices. In addition, any hyperedge in E can be grouped into two disjoint subsets, called a predecessor set and successor set. If the predecessor set is included in V^i, the successor set will be included in V^{i+1}. The depth of a vertex v_j is denoted as $D(v_j)$. If $v_j \in V^i$, $D(v_j) = i$. An example shown in Fig. 3.2 gives a three-depth hypergraph.

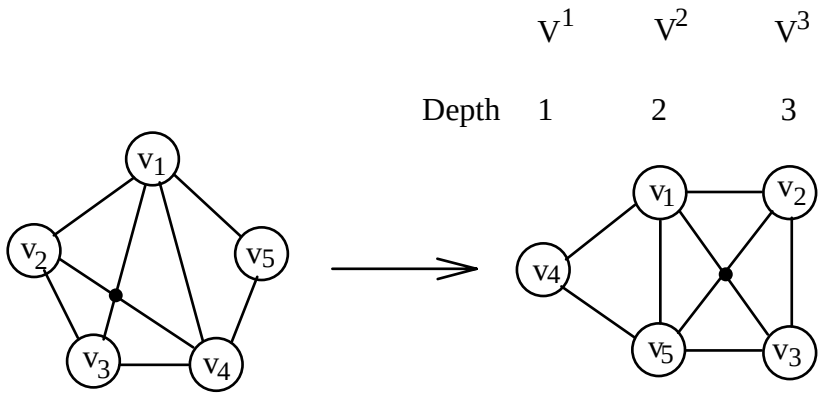


Fig. 3.2 A hypergraph and vertex depth.

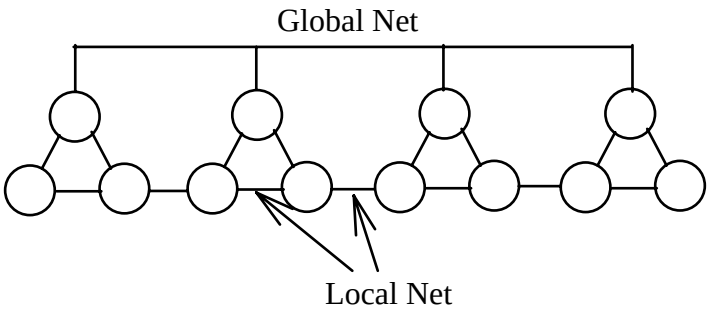


Fig. 3.3 Global nets versus local nets

The procedure used to decide vertex depth is given in the following.

```
procedure VERTEX_DEPTH(H)
/*   Input: H=(V,E)*/
/*Output: A sorted vertex list V̄ with vertex depth in nondecreasing order.*/
/*V̄ =<V^1, V^2, ..., V^m>*/
v_in ← v_1
for i ← 2 to |V| do
      if |E(v_i)| < |E(v_in)| then v_in ← v_i endif
repeat
(D(v_in), i, V^1) ← (1, 1, {v_in})
while (i < m) do
      D(v_in) ← D(v_in) + 1
      i←i+1
      V^i ← V(E(v_in))
      v_in = v_in ∪(V(E(v_in))
repeat
OUTPUT(V̄)
```

end VERTEX_DEPTH

If the size of a hyperedge is too large, it is called a **global hyperedge**, and it will be deleted from the hypergraph at this stage. An example depicted in Fig. 3.3 shows the difference between global and local hyperedges. Intuitively, global hyperedges interconnect the local clusters of the hypergraph and tend to be associated with many vertices. On the other hand, local hyperedges are internal to clusters. It is obvious that global hyperedges will destroy the ordering of vertices; thus, we have to delete them from the hypergraph. In practice, the difference between global and local hyperedges may not be obvious in a hypergraph. In this study, we differentiate global and local hyperedge in its size. According to our experience, the critical size of global hyperedges is in the range from 5 to 11. Sometimes it is necessary to try a few different parameters to find the best solution.

3.1.1.2 **The Procedure of the Seed Generator**

If a hypergraph is to be divided into k parts, k seeds need to be selected. In the seed selection phase, vertices are first divided into k of equal size parts according to their depth. Then, for each part, the most heavily connected vertex is chosen as a seed.

The details of the procedure are as follows.

procedure SEED_GENERATOR(H, n, C)
/* Input : H and the number of parts n*/
/* Output: the set of seeds, C.*/
$\overline{V} \leftarrow$VERTEX_DEPTH(H)
if m mod n $>$ 0.5×[m/n] **then** d $\leftarrow$ [M/n] + 1
else d $\leftarrow$ [M/k]
endif /* [ ] represents the Gaussian function*/
for i $\leftarrow$ 1 **to** m **do**
n $\leftarrow$ [(i-1)/d] + 1
if $|E(v_x)| > |E(c_n)|$ **then** $c_n = v_x$
repeat
end SEED_GENERATOR

3.1.2 **Function SELECT_ATOM(A)** :

The function selects the candidate mainly based on an evaluation function, called **inside-outside connectivity** (**IOC)**, which measures the net connectivity of the atom considered between the set of clusters and the set of other atoms. For an atom, a_t, $E(a_t)$ can be divided into three disjoint subsets of $E_{in}(a_t)$, $E_{pi}(a_t)$, and $E_{ex}(a_t)$, called the set of internal hyperedges, the set of partial internal hyperedges, and the set of external hyperedges, respectively. They are defined by

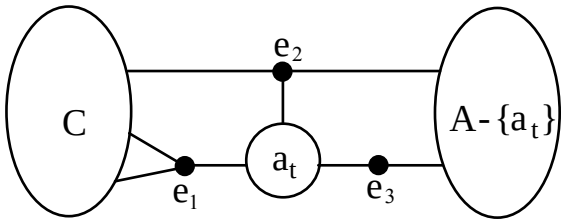


Fig. 3.4 Three types of hyperedges.

$E_{in}(a_t) = \{e_d \mid e_d \in E(a_t), V(e_d) - \{a_t\} \subseteq C\}$
$E_{ex}(a_t) = \{e_d \mid e_d \in E(a_t), V(e_d) \subseteq A\}$
$E_{pi}(a_t) = E(a_t) - E_{in}(a_t) - E_{ex}(a_t)$.

An example is depicted in Fig. 3.4, where e_1, e_2 and e_3 belong to $E_{in}(a_t)$, $E_{pi}(a_t)$, and $E_{ex}(a_t)$, respectively. $IOC(a_t)$ is given by

$$IOC(a_t) = |E_{in}(a_t)| - |E_{ex}(a_t)| \quad .$$

The contribution of $E_{pi}(a_t)$ is not contained in the definition of IOC since each hyperedge in $E_{pi}(a_t)$ links both sets of clusters and other atoms.

The atom with the highest IOC apparently has higher net connectivity with the set of clusters than the set of other atoms. Thus this atom is prior to be selected as a candidate. The rules to select the candidate are:

1) The atom which neighbors only one unsaturated cluster is chosen. If tie,
2) The atom of the largest IOC is chosen. If tie again,
3) The atom which neighbors fewer clusters is chosen.

The application of rule 1 is demonstrated for a special case that an atom has only one neighbor that is an unsaturated cluster such as shown in Fig. 3.5. In this case, a_1 and c_1 are **locked** into each other since a_1 apparently belongs to c_1.

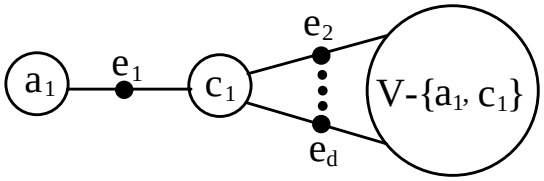


Fig. 3.5 The a_1 and the c_1 are locked each other.

3.1.3 **Function SELECT_CLUSTER(a_c, C)**:

If a cluster is locked with a_c, it is naturally chosen as the merged cluster. Otherwise, the merged cluster is chosen from the unsaturated clusters that neighbor to a_c mainly based on an objective function, named **contribution value** (CV), that measures the contribution of deleting hyperedges from the hypergraph if a_c is assigned into this cluster. For defining it, $E(a_c)$ is divided into two disjoint sets, called the set of **uncut hyperedges**, $E_u(a_c)$, and the set of **cut hyperedges**, $E_c(a_c)$. A hyperedge, $e_d \in E(a_c)$, is cut if it is incident on more than one cluster. In addition, even if e_d is incident on one cluster, c_ℓ, it is also cut if $|V(e_d)|$-1 is larger than the number of atoms that c_ℓ can have, because of c_ℓ can not include all atoms inside $V(e_d)$. In other cases, the hyperedge is uncut. The formal definition is:

$E_c(a_c)=\{e_d | e_d \in E(a_c), |V(e_d) \cap C| > 1 \vee [V(e_d) \cap C = \{c_\ell\} \wedge |V(e_d)| > C_p(c_\ell) + 1]\}$.
$E_u(a_c) = E(a_c) - E_c(a_c)$.

where $C_p(c_\ell)$ is the number of atoms that c_ℓ can include further.

If a_c is assigned into the merged cluster c_m, the hyperedges in $E_c(a_c)$ cannot be deleted from the hypergraph and are remained in the final shunk hypergraph; the uncut hyperedges which include only a_c and c_m are deleted from the hypergraph; the uncut hyperedges that contain not only a_c and c_m but also other atoms cannot be deleted from the hypergraph but is contracted to one-vertex-less hyperedges.

The $CV(a_c, c_m)$ then is defined by

$$CV(a_c, c_m) = \sum_{e_d \in E_u(a_c)} \frac{1}{|V(e_d)| - 1}$$

Since the aim in the clustering algorithm is to partition a hypergraph into several sub-hypergraphs (clusters) with the min-cut objective, the rules to select the merged cluster then are:

1) The cluster that is locked with a_c is selected. If tie,
2) The cluster of the largest CV is chosen. If tie again,
3) The cluster which can include more atoms is chosen.

3.1.4 Function ASSIGN(a_c, H_m)

This function assigns a_c and $E(a_c)$ into the sub-hypergraph, H_m, by the following procedure.

procedure ASSIGN(a_c, H_m)
$V_m \leftarrow V_m + \{a_c\}$
for each $e_d \in E(a_c)$ **do**
  if $e_d \in E_{ex}(a_c)$ **then** [$V(e_{d,m}) \leftarrow \{a_c\}$; $E_m \leftarrow E_m + \{e_{d,m}\}$]
    else $V(e_{d,m}) \leftarrow V(e_{d,m}) + \{a_c\}$
  endif
r**epeat**
end ASSIGN

3.1.5 **Function HYPERGRAPH_CONTRACTION(a_c, c_m, H)**

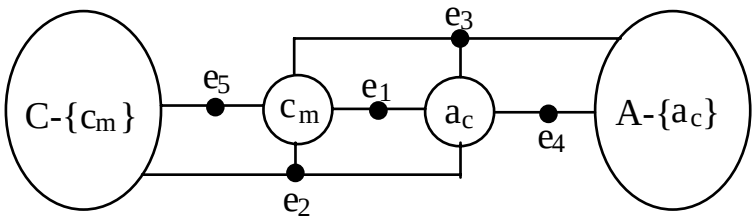
The previous hypergraph is reduced to a one-vertex-less hypergraph by the following steps in this function.

procedure HYPERGRAPH_CONTRACTION(a_c, c_m, H)
$V \leftarrow V - \{a_c\}$; $E \leftarrow E - [E_{in}(a_c) \cap E_u(a_c)]$
for each $e_d \in E(a_c)$ **do** $V(e_d) \leftarrow [V(e_d) - \{a_c\}] \cup \{c_m\}$ **repeat**
end HYPERGRAPH_CONTRACTION

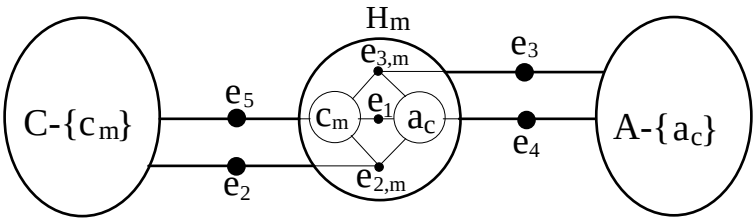
Figs 3.6(a) and (b) show an example to explain the change of a hypergraph after one assignment process, where a_c is assigned into H_m. The internal and uncut hyperedge, e_1, is assigned into H_m and deleted from the primary hypergraph; the uncut hyperedges, e_2 and e_3 are partially assigned into H_m and both are reduced to one-size-less hyperedges; e_4 and e_5 are still left in the reduced hypergraph and a_c is replaced by c_m in $V(e_4)$.

3.2 **Time Complexity Analysis**

Although the algorithm can be used to divide a hypergraph into multiple parts, for simplifying the complexity analysis only a two-way partitioning of a hypergraph is discussed.



(a) The previous hypergraph. (continue)



(b) The reduced hypergraph after assigning a_c into H_m.

Fig. 3.6 An example to explain one clustering pass.

3.2.1 **Space Complexity**

Basically, the procedure of the present algorithm is a sequence of cluster tree construction and hypergraph reduction. Thus, the memory space needed is the sum of the size of the initial hypergraph and the final cluster tree. So, the space complexity is O(M).

3.2.2 **Time Complexity**

For a hypergraph, H=(V, E), let m and n are the numbers of vertices and hyperedges, respectively. Two measures are used to analyze the time complexity of the present algorithm. The first is the size of the hypergraph (referred to as the global complexity), which can be defined by m+n. The second is the degree of the cluster c_j, written by $|e(c_j)|$, (called the local complexity), which measures the complexity of combining an atom with a cluster. As the merging passes proceeds, the number of general vertices in the hypergraph decreases, but the size of clusters increases. That is, the global complexity decreases, but the local complexity increases. Here, both types of complexities have to be considered.

3.2.2.1 **Power-Law Relationship**

By applying Rent's rule, the measure of the degree of a cluster can be replaced by its size. **Rent's Rule** has the form of

$$|e(c_i)| = p \times |c_i|^r$$

where p is the average degree of the atoms inside the cluster c_i, and r is the Rent exponent. Different clusters may have different values of p and r, but in view of asymptotic analysis, the difference can be neglected if r and p are replaced by their maximum values. Thus, to simplify the analysis, we assume p and r to be constants in time complexity analysis.

3.2.2.2 **Time Complexity of Seed Generator**

Critical step of seed generator is to decide the vertex depth. This step begins with the selection of an initial vertex. This requires O(m) time. Later, each vertex will be processed once and merged with the initial vertex. For each merging pass, we have to compare the hyperedges incident on the initial vertex to those incident on the merged vertex. The operator requires $O(k^r)$ time, where k is the size of the initial vertex. There are M-1 merged passes; thus, the entire time complexity is

$$\sum_{k=2}^{m} O(k^r) \sim O(m^{r+1})$$

3.2.2.3 **Time Complexity of Cluster Growth Algorithm**

The forest of two binary trees to represent the hierarchical structure of merging processes is depicted in Fig. 3.7. Each internal node represents a merging pass. A numeral in a circle denotes the size of the cluster. Considering the merged pass at level m/2 - k + 1 in Fig. 3.7, the cluster has size k and degree $p \times k^r$, and the candidate vertex has size 1 and degree p.

For each merging pass, selecting the candidate vertex requires no more than O(n) time, and selecting the merged cluster requires only O(1) time. The procedure for joining the list of the merged cluster and the list of a candidate vertex requires $O(k^r)$ time.

The above process must to be repeated until all general vertices have been assigned to clusters. Thus, the time complexity is

$$\sum_{k=2}^{m} O(n) + 2 \times \sum_{k=2}^{m/2} O(k^r) \sim O(m^{r+1} + m \times n)$$

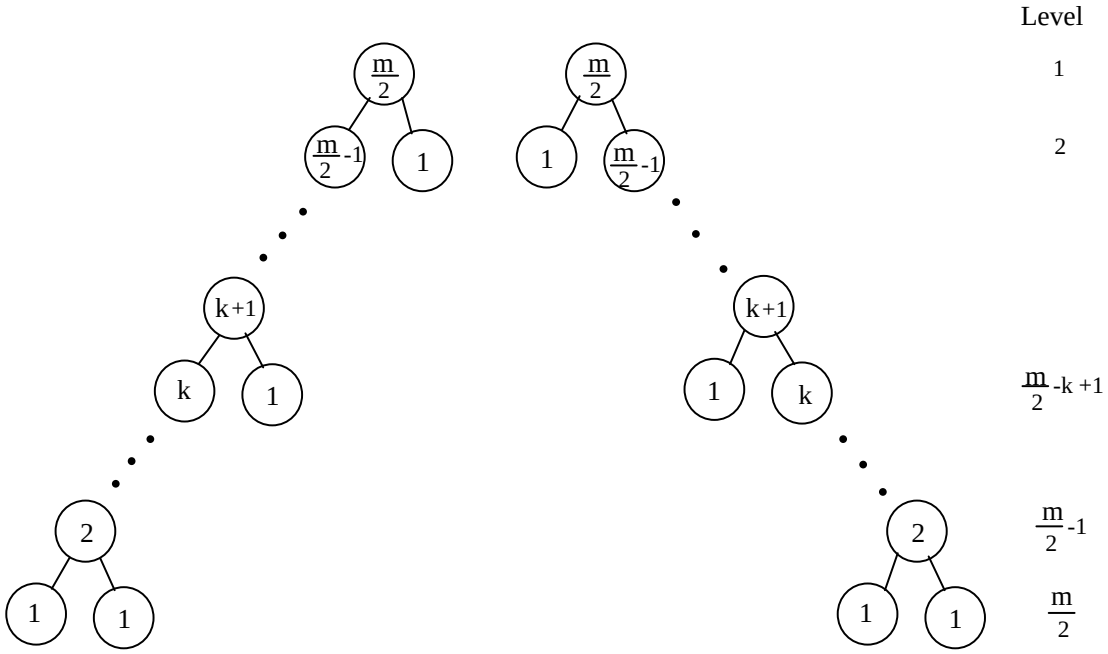


Fig. 3.7 Tree structure of vertex combination process. The number in a node denote the size of the cluster tree with root of the node.

3.2.2.4 **Total Time Complexity**

From the above analysis, one can conclude that the total time complexity of the present algorithm is $O(m \times n + m^{r+1})$. In general, M and N are of the same order, and $0.47 \leq r \leq 0.75$ [Lan71], so the worst case time complexity of present algorithms $O(M^2)$.

3.3 **Experimental Results and Discussion**

The clustering algorithm, including the hypergraph model, graph model and weighted graph models, has been implemented in C language and run on a SUN SPARK I workstation.

In order to compare the performance difference among those algorithms, a simple example, shown in Fig. 3.8, was examined. This example included fourteen vertices and six nets. The distributions of net size were 3 two-pin nets, a six-pin net, a seven-pin net and a nine-pin net. The requirement here was to partition the network into two parts of equivalent vertices. c_1 and c_2 were chosen as seeds. Five clustering algorithms, one based on hypergraph model, another based on graph model and three others based on weighted graph models with different weights, were examined. The resulting distribution of vertex per net is shown in Table 3.1. There existed an optimum solution to partition the network into two parts with only two connecting nets. The optimum solution is shown in Fig. 3.8(a) and column 8 in Table 3.1.

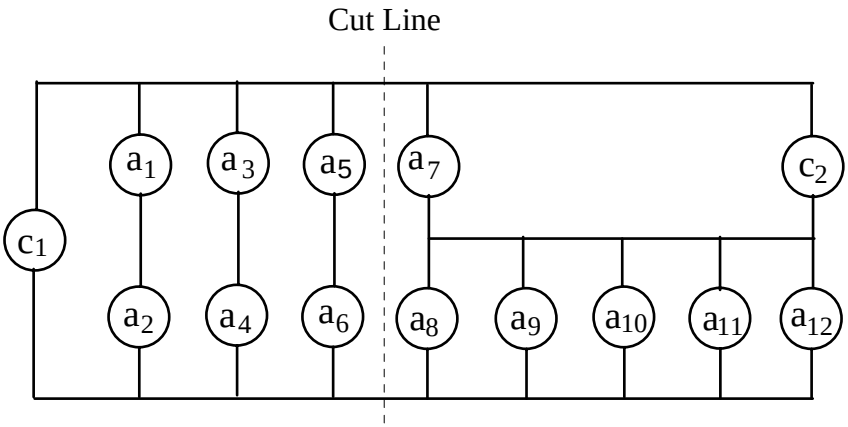
Fig. 3.8(b) and column 3 in Table 3.1 show the partitioning results from the graph model. Obviously, it is a worst-case solution, because the cut-line cuts across all the nets. Fig. 3.8(b) also represents the ordering of cluster growth, and the numerals denote the merging order. It can be seen that the growth path was along the large size net. This phenomenon shows that this algorithm devours larger size nets before smaller size ones. It is inefficient, and always leads to bad solution, since it cuts through too many smaller size nets. An edge weighting of less than 1 could break the situation of growing along large size nets.

Fig. 3.8(c) and Fig. 3.8(d) show the results using edge weights of 2/(n-1) and 2/n, respectively. The corresponding distribution of vertex per net is shown in column 4 and 5 in Table 3.2. In both cases, only three nets were cut and no two-pin nets were cut. These solutions are better than that from the graph model, but are worse than those from the hypergraph model and the weighted graph model with an edge weight of $4/(n^2\text{-mod}(n,2))$. The latter two give the best solution as depicted in Fig. 3.8(e) and Fig. 3.8(f), respectively.

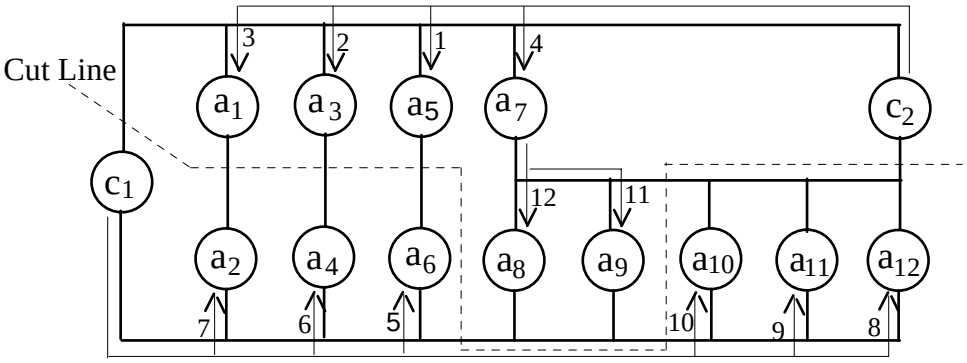
To evaluate the performance difference further, a more complicated example and its partitioning results are given in Table 3.2. This example included 106 components and 68 nets. The range of net size was from 2 to 21. The partitioning results for the graph model are shown in the third column in Table 3.2. As expected, few of the nets with high pin counts (above 4, here) were divided, while significantly more three-pin nets were divided. On the contrary, the results for the hypergraph model shown in seventh column in Table 3.2 divided the nets with high pin counts (above 9, here), and only a two-pin net and 2 three-pin nets were divided. Between these two extremes are the results from the weighted graph models.

Table 3.1 The comparison among different graph models for Network 1.

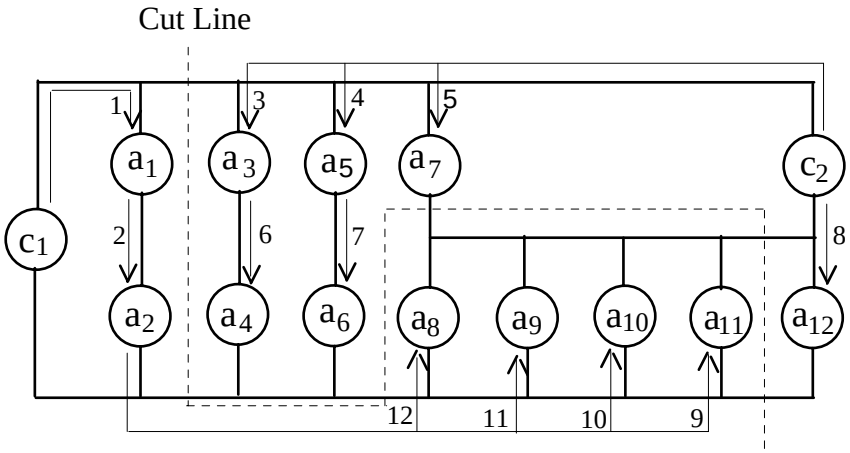
Pins per Net	Number of Nets	Graph	Weighted Graph			Hypergraph	Optimum Solution
			W=2/(n-1)	W=2/n	W=4/(n^2-n%2)		
2	3	3	0	0	0	0	0
6	1	1	1	1	1	1	1
7	1	1	1	1	0	0	0
9	1	1	1	1	1	1	1
Totals	6	6	3	3	2	2	2



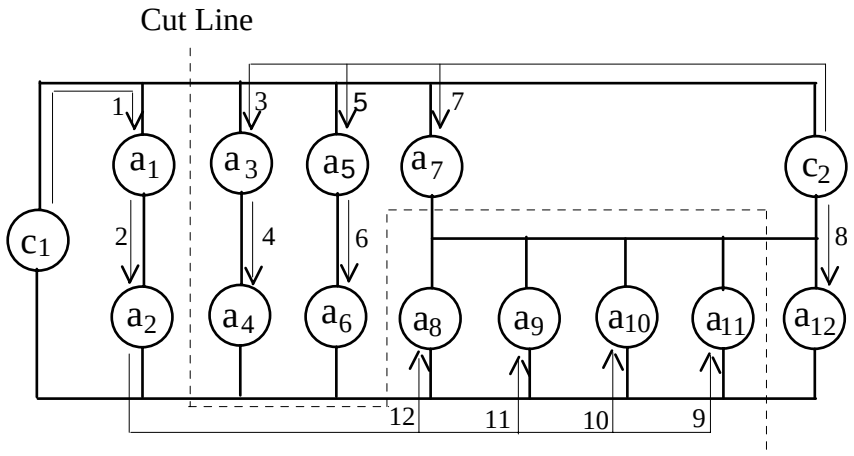
(a) The network and optimum solution of a two-way partition.



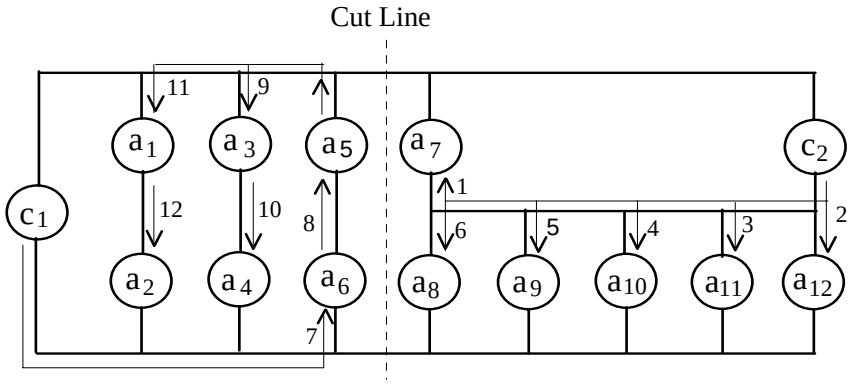
(b) The cluster growth procedure based on a graph model.
(continue)



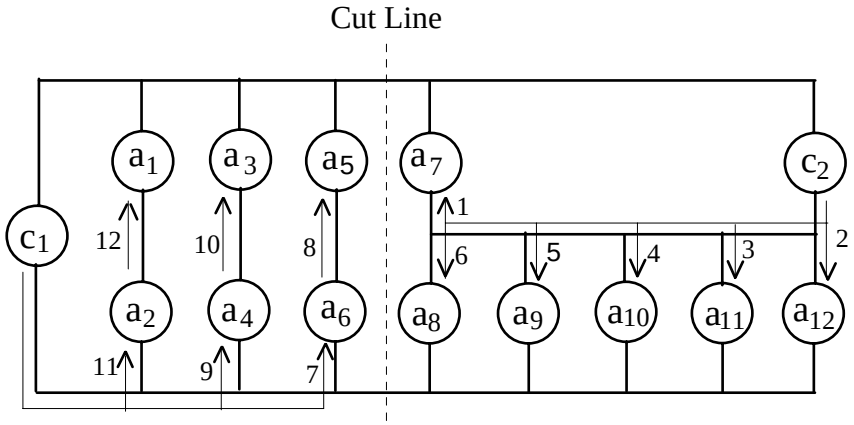
(c) The cluster growth procedure based on a weighted graph model of 2/(n-1) edge weighting.



(d) The cluster growth procedure based on a weighted graph model of 2/n edge weighting.



(e) The cluster growth procedure based on a weighted graph model of $4/[n^2\text{-mod}(n,2)]$ edge weighting.



(f) The cluster growth procedure based on a hypergraph model.

Fig. 3.8 A network example. A numeral beside an edge denotes the ordering of merging the pointed vertex.

Table 3.2 **The comparison among different graph models for Network 13.**

Pins per Net	Number of Nets	Graph	Weighted Graph			Hypergraph
			W=2/(n-1)	W=2/n	W=4/(n^2-n%2)	
2	20	1	1	1	0	1
3	36	21	10	12	12	2
4	4	1	2	0	2	0
5	1	0	0	1	0	0
6	1	0	1	1	0	0
9	2	0	2	2	2	2
10	2	1	2	2	2	2
13	1	0	1	1	0	1
21	2	0	1	1	1	1
Totals	68	24	20	21	19	9

Table 3.3 **Statistics of examined networks.**

	Cells	Nets	Cell-Net Connections	Range of Net Sizes
Network 1	14	6	28	2~9
Network 2	31	33	81	2~6
Network 3	44	26	101	2~11
Network 4	56	36	128	2~8
Network 5	65	36	132	2~13
Network 6	42	37	142	2~19
Network 7	61	38	144	2~19
Network 8	48	48	148	3~4
Network 9	**77**	53	172	2~11
Network 10	91	54	197	2~24
Network 11	92	62	209	2~18
Network 12	85	96	256	2~10
Network 13	106	68	268	2~21
Network 14	294	186	540	2~16
Network 15	533	313	943	2~24
Network 16	524	308	944	2~22
Network 17	614	374	1112	2~18
Network 18	674	415	1192	2~22
Network 19	786	489	1408	2~20
Network 20	1453	922	2676	2~20

Table 3.4 **The crossing counts of networks in Table 3.3 after partitioning.**

Networks	Graph	W = 2/(n-1)		W = 2/n		W = 2/(n^2-n%2)		Hypergraph	
	Crossing Count	Crossing Count	Reduction (%)	Crossing Count	Reduction (%)	Crossing Count	Reduction (%)	Crossing Count	Reduction (%)
1	6	3	50	3	50	2	66.6	2	66.6
2	7	6	14.3	6	14.3	5	28.8	4	42.9
3	12	6	50	8	33.3	9	25	5	58.3
4	21	9	57.1	8	61.9	12	42.9	6	71.4
5	10	11	-10	10	0	11	-10	8	20
6	18	8	55.6	8	55.6	8	55.6	4	77.8
7	22	10	54.5	9	59.1	6	72.7	7	68.2
8	25	18	28	16	36	16	36	12	52
9	13	14	-7.7	13	0	10	23.1	5	61.5
10	19	10	47.4	6	68.4	13	31.6	8	57.8
11	19	15	21	15	21	16	15.8	10	47.4
12	32	25	21.9	23	28.1	22	31.3	20	37.5
13	24	20	16.7	21	12.5	19	20.8	9	62.5
14	21	20	4.8	20	4.8	14	33.3	10	52.4
15	14	15	-7.1	16	-14.3	17	-21.4	10	28.6
16	34	33	2.9	25	26.5	18	47.1	14	58.8
17	18	28	-55.5	22	-22.2	25	-38.9	15	16.7
18	28	31	-10.7	28	0	14	50	14	50
19	37	31	16.2	33	10.8	17	54.1	17	54.1
20	37	33	10.8	27	27.0	29	21.6	17	54.1
Summation	417	346	17.0	317	24.0	283	32.1	197	52.8

We have examined all the examples listed in Table 3.3. The results for crossing count are listed in Table 3.4. Note that the hypergraph model is better than the graph model in all cases and is always better than the weighted graph models. From Table 3.4, we can conclude that the algorithm based on the hypergraph model can reduce crossing counts by about 53% than the algorithm based on the graph model, and that those algorithms based on the weighted graph models only reduced crossing counts by about 17% ~ 32%. The definition about **reduction** is

$$\frac{\text{(Crossing count from the graph model)-(Crossing count from the compared model)}}{\text{(Crossing count from the graph model)}}$$

Running time comparisons are shown in Fig. 3.9. The running time of the hypergraph model is proportional to $M^{1.73}$, where M is the vertex number of the hypergraph. The runnung times of both graph and weighted graph models are proportional to $M^{1.41}$, but the weighted graph models take about 11% more time than does the graph model.

Fig. 3.9 Running time against vertex number.

3.4 **Summary**

Traditional clustering algorithms usually use a graph structure to model electrical networks. However, because of edge exaggeration, the graph model cannot exactly represent a net that interconnects more than two electrical components. The transformation from an electrical network to a graph usually induces some traps, and leads the clustering procedure to grow along the nets of large size. Previous authors usually used weighting functions to modify edge exaggeration. This strategy can decrease the intensity of traps from nets of large size but always makes new traps from small-sized nets.

This paper describes a new clustering algorithm, which is based on a hypergraph model. It is different from previous algorithms, which used either graph models or weighted graph models. The hypergraph model can exactly represent electrical networks, and no distortion is induced in the transformation from the network to its corresponding hypergraph. Computational results show that the crossing count obtained by the present algorithm is as much as 53% better than the result from graph model of unity weight, and about 22% ~ 38% better than the results from weighted graph models. Complexity analysis proves that the present algorithm requires $O(M^2)$ time, where M is the vertex number of a hypergraph. Running time testing shows that the algorithm requires $O(M^{1.73})$ time.

Though the new clustering algorithms is presented for the partitioning problems, it also can be use to solve the linear permutation problems that will be described in the last chapter, if the algorithm is executed from one seed [Kan83].